



Let's Swift – WRITE THE CODE. CHANGE THE WORLD.

首页
关于Swift
Swift资讯
Swift教程
Swift实战
Swift代码库

FOLLOW:



SWIFT实战

Q 1

NEXT STORY

iPhone 6 以及 iPhone 6 Plus 正式发布



在Swift中自定义下标 (Subscripts)

BY 史薇美特 · 2014年9月5日 · 阅读量: 2,698

通常情况下，我们在使用数组 (Array) 或字典 (Dictionary) 时会使用到下标。其实在Swift中，我们还可以给类 (class) 自定义下标，下面就让我们来看看Swift中是如何自定义下标的。

通过Subscript赋值和获取值

我们先看看下面这个类：

```
class DailyMeal
{
    enum MealTime
    {
        case Breakfast
        case Lunch
        case Dinner
    }

    var meals: [MealTime : String] = [:]
}
```

我们使用该类时可以直接用 meals 字典，以枚举作为key来查询，像这样：

```
var monday = DailyMeal()

monday.meals[.Breakfast] = "Toast"

if let someMeal = monday.meals[.Breakfast]
{
    println(someMeal)
}
```

到目前呢，我们创建了 DailyMeal 类的实例变量 monday，并可以使用 DailyMeal 类中的 meals 字典进行查询。但是大家有没有感觉 monday.meals[] 这种写法很累赘呢？至少我看 meals 就很不顺眼，有没有更简单快捷的方法让我们直接使用 monday 变量就能赋值或获取 Breakfast 的值呢？别着急，今天的主角要登场了，让我们先在 DailyMeal 类中添加如下代码：

```
subscript(requestedMeal : MealTime) -> String?
{
    get
```

分类目录

Swift代码库

Swift实战

Swift教程

Swift视频教程

Swift语言参考

Swift语言基础

Swift资讯

关于Swift

标签

API app apple watch Cocoa Objective-C OC
Swift text UI uikit watch watchkit 下标
代码库 函数 初始化 动力 基础 字典 字符
串 实例 实战 属性 感叹号 扩展 教程 数组
方法 构造 析构 枚举 概览 源码 看法 程序员
简介 类 结构 继承 编程 设计模式 语句 语
言参考 闭包 问号

特别提示
查看标识获取更多信息



在Swift中自定义下标（Subscripts） | Let's Swift

```
{
    return meals[requestedMeal]
}
set(newMealName)
{
    meals[requestedMeal] = newMealName
}
}
```

上面的代码就是一个自定义下标，看起来是不是有点像计算类属性的 getter、setter 方法的写法呢？但是它们还是有区别的，首先下标使用 subscript 关键字，然后跟一个圆括号，里面是该下标的参数和参数类型（在实际使用中该参数就相当于数组的 index 和字典的 key 一样），最后有该下标的返回值类型。

从上面代码可以看到，在下标的 getter 和 setter 方法中，其实还是在对 meals 数组进行操作，但是我们通过下标就可以将对 meals 数组的操作屏蔽掉。现在来看看我们应该怎样使用：

```
var monday = DailyMeal()

monday[.Breakfast] = "Toast"

if let someMeal = monday[.Breakfast]
{
    println(someMeal) // Toast
}
```

现在是不是已经没有碍眼的 meals 了呢，使用起来更加简洁，语义也更加明确了呢，这就是下标最简单的一个用法。

如果上面的代码中我们不给 monday[.Breakfast] 赋值，直接输出值会得到什么结果呢？细心的同学可能会注意到，在定义下标时它的返回值是 Optional 类型的，所以不赋值直接输出的结果是 nil，这样就显得太没礼貌了，所以我们再来改造一下下标的代码：

```
subscript(requestedMeal : MealTime) -> String
{
    get
    {
        if let thisMeal = meals[requestedMeal]
        {
            return thisMeal
        }
        else
        {
            return "Ramen"
        }
    }
    set(newMealName)
    {
        meals[requestedMeal] = newMealName
    }
}
```

我们看到下标的返回值从 String? 改为了 String，那么相应的我们要在 getter 方法中对 meals[requestedMeal] 的值进行判断，如果没有赋值的话，我们将返回一个默认值 Ramen（兰州拉面让人欲罢不能）。现在我们就可以这样用啦：

```
var monday = DailyMeal()

monday[.Lunch] = "Pizza"

println(monday[.Lunch])           //Output: "Pizza"
```

```
println(monday[.Dinner]) //Output: "Ramen"
```

现在使用 `DailyMeal` 类是不是感觉到很简介，语义很明确也很健壮呢，答案是肯定的。我们通过下标避免向用户暴露不必要的API，同时也达到了高维护性的目的。

`DailyMeal` 类的完整代码如下：

```
class DailyMeal
{
    enum MealTime
    {
        case Breakfast
        case Lunch
        case Dinner
    }

    var meals: [MealTime : String] = [:]

    subscript(requestedMeal : MealTime) -> String
    {
        get
        {
            if let thisMeal = meals[requestedMeal]
            {
                return thisMeal
            }
            else
            {
                return "Ramen"
            }
        }
        set(newMealName)
        {
            meals[requestedMeal] = newMealName
        }
    }
}
```

只读下标

何为只读下标，顾名思义就是不能通过下标赋值，只能通过下标查询。这种下标的应用场景一般是实现一些数据公式、数据函数，它们一般都是只需要你指定一个数字，然后返回该公式对该数字的计算结果。下面我们用一个阶乘的例子来说明只读下标：

```
struct FactorialGenerator
{
    subscript(n: Int) -> Int
    {
        var result = 1

        if n > 0
        {
            for value in 1...n
            {
                result *= value
            }
        }

        return result
    }
}
```

同学们可能已经注意到了，上面的下标并没有 `getter` 和 `setter` 方法。这是因为，如果你想定义一个只读的下标，那么可以不实现 `setter` 方法，并且可以省略 `getter` 方法的 `get` 关键字。Swift的编译器会判断出这是一个只读的下标，如果

在Swift中自定义下标（Subscripts） | Let's Swift
你强行通过下标赋值，那么编译器会报错。

让我们来使用以下这个阶乘结构体：

```
let factorial = FactorialGenerator()

println("Five factorial is equal to \(factorial[5]).")
//Output: "Five factorial is equal to 120."

println("Ten Factorial is equal to \(factorial[10]).")
//Output: "Ten Factorial is equal to 3628800."
```

当然上面这个示例只是展示了只读下标的语法和应用场景，阶乘的实现逻辑在这就不累赘了。总的来说，我们可以通过下标简化暴露给用户的API，你可以在用户毫不知情的情况下更改某个API的功能。不仅使代码更易读，同时也大大提高了代码的可维护性，是不是很酷呢！

参考原文：Custom Subscripts in Swift，中文译文原文：
<http://www.devtalking.com/articles/custom-subscripts-in-swift/>

0



app开发报价单



ios培训班



前端学习路线



前端工程师待遇



程序员培训

python闭包

单点登录原理

ios项目

软件工程师培训

python 闭包

游戏设计

静态语言

单点登录demo

ui设计培训

ui设计培训机构

ios技术

阿迪正品折扣店

pyt



前端工程师待遇



app开发报价单



玻璃钢电缆支架



自学ui设计



婚礼开场视频

Tags: 下标 字典 数组

YOU MAY ALSO LIKE...



Swift UI开发初探 (二)
UI创建和方法调用

17 六, 2014



iOS8 Day-by-Day- Day2
一 分享应用扩展

24 九, 2014




在Swift中使用C语言的指
针

3 八, 2014

1 RESPONSE

Comments 1 Pingbacks 0

 Ying 2016年5月12日上午7:17
非常棒的网站,今天才在无意间看到
回复

发表评论

姓名 *

电子邮件 *

站点

评论

发表评论

查看标识获取更多信息

标签

API app apple watch Cocoa Objective-C OC
Swift text UI uikit watch watchkit 下标 代码
库 函数 初始化 动力 基础 字典 字符串 实例
实战 属性 感叹号 扩展 教程 数组 方法 构造
析构 枚举 概览 源码 看法 程序员 简介 类 结构
继承 编程 设计模式 语句 语言参考 闭包
问号

分类目录

- Swift代码库
- Swift实战
- Swift教程
- Swift视频教程
- Swift语言参考
- Swift语言基础
- Swift资讯
- 关于Swift

关于我

小组微



联系我们：letsswift@163.com

Swift站点导航

Swift小组



友情链接：SwiftV课堂 | SwiftChina | HelloSwift | Swiftist社区 | 9ria游戏开
发者社区 | 刚刚在线 | 智慧寺院 | 智慧寺院论坛 | 佛教建筑网 | 洛克威尔咨
询 | 程序员之家

京ICP备14018534号-2

Let's Swift © 2016. All Rights Reserved.

Powered by WordPress. Theme by Alx. 

